A Multilevel Algorithm for Large Unconstrained Binary Quadratic Optimization

Yang Wang[1], Zhipeng Lü[2], Fred Glover[3], and Jin-Kao Hao[1]

[1] LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
[2] School of Computer Science and Technology, Huazhong University of Science and Technology, 430074 Wuhan, China
[3] OptTek Systems, Inc., 2241 17th Street Boulder, CO 80302, USA
{yangw,hao}@info.univ-angers.fr, zhipeng.lv@hust.edu.cn, glover@opttek.com

Abstract. The unconstrained binary quadratic programming (UBQP) problem is a general NP-hard problem with various applications. In this paper, we present a multilevel algorithm designed to approximate large UBQP instances. The proposed multilevel algorithm is composed of a backbone-based coarsening phase, an asymmetric uncoarsening phase and a memetic refinement phase, where the backbone-based procedure and the memetic refinement procedure make use of tabu search to obtain improved solutions. Evaluated on a set of 11 largest instances from the literature (with 5000 to 7000 variables), the proposed algorithm proves to be able to attain all the best known values with a computing effort less than any existing approach.

Keywords: Multilevel approach, unconstrained binary quadratic optimization, hybrid method, memetic algorithm, tabu search.

1 Introduction

The objective of the unconstrained binary quadratic programming (UBQP) problem is to maximize the function:

$$f(x) = x'Qx = \sum_{i=1}^{n} Q(i,i) \cdot x_i + \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} Q(i,j) \cdot x_i \cdot x_j \quad (1)$$

where $Q = [Q(i,j)]$ is an n by n symmetric matrix of constants and x is an n-vector of binary (zero-one) variables, i.e., $x_i \in \{0,1\}$, $i = 1, \ldots, n$. (Considering that a general item of $f(x)$ is $(Q(i,j)+Q(j,i)) \cdot x_i \cdot x_j$, we set $Q(i,j) = Q(i,j) + Q(j,i)$ and $Q(j,i) = 0$, $(i < j)$ to simplify the coefficient of $x_i \cdot x_j$.)

UBQP is a well-known NP-hard problem that can formulate numerous applications in diverse areas, such as those from financial analysis [23], social psychology [14], machine scheduling [1], computer aided design [19] and cellular radio channel allocation [9]. Moreover, it is a unified model for a variety of combinatorial optimization problems, such as graph coloring problem, maxcut problem,

N. Beldiceanu, N. Jussien, and É. Pinson (Eds.): CPAIOR 2012, LNCS 7298, pp. 395–408, 2012.

set packing problem, etc. These problems can be easily recast into the form of UBQP, and then solved by applying any UBQP algorithm. More information can be found in [18] for the general transformation procedures.

Due to its theoretical significance as an NP-hard problem and its immense potential applications, UBQP has attracted researchers to design various solution procedures to tackle it. Exact methods based on branch and bound or branch and cut [7,15,29] are quite useful to obtain optimal solutions to instances of limited sizes. However, because of the high computational complexity, heuristic and metaheuristic algorithms are commonly used to create approximate solutions to larger problem instances. Examples of these methods include local search [8], simulated annealing [4,16], tabu search [11,13,27,28,32,33], scatter search [2], evolutionary and memetic algorithms [6,20,21,25,26], and neural network algorithm [34].

These algorithms have continually improved our capability to find satisfactory solutions to many problem instances. However, we observe that many metaheuristic UBQP algorithms encounter difficulties when they are applied to large instances (with more than 5000 variables) .

In this work, we are interested in investigating the so-called multilevel approach to handling large UBQP instances. The multilevel approach is known to be useful to tackle large instances of several other types of combinatorial optimization problems [36]. For example, multilevel algorithms are among the best performing approaches for large graph partitioning problems [31,35,24,5].

Generally, the multilevel paradigm consists of three phases [36]: (1) a coarsening phase to create a hierarchy of coarser (smaller and intermediate) problems through grouping or extracting problem variables; (2) an initial optimization phase to obtain a solution to the coarsest (smallest) problem using an optimization procedure; (3) an uncoarsening phase (also called projection) to recover progressively each intermediate problem and apply to it the optimization procedure to further improve the solution quality.

In this paper, we investigate for the first time the multilevel approach applied to UBQP. The proposed multilevel algorithm integrates a coarsening phase based on the backbone notion [32] (Section 2.2), a population-based memetic optimization procedure (Section 2.4) and an asymmetric uncoarsening phase (Section 2.5). Experiments on a set of 11 largest UBQP benchmark instances from the literature demonstrate that our proposed algorithm is able to attain the current best known results with much less computing time than any other existing algorithm (Section 3).

2 The Backbone Multilevel Memetic Algorithm

2.1 The General Multilevel Scheme

The general scheme of our multilevel algorithm for UBQP is shown in Algorithm 1. To begin with, the initial matrix Q_0 is transformed into a sequence of coarser matrices $Q_1, \ldots, Q_q$ such that $n_1 > \ldots > n_q$ where each n_i $(i = 1, \ldots, q)$ is the number of variables in Q_i. To obtain each intermediate matrix, we apply

the idea of a backbone to create and extract backbone variables, as explained in Section 2.2). This coarsening phase stops when q reaches a prefixed value called the threshold level. For the series of matrices $Q_0, \ldots, Q_q$, we call Q_0 the highest level problem and Q_q the lowest level problem.

The next phase aims to generate an initial (optimized) solution to the lowest level problem Q_q. In our case, we employ the population-based hybrid metaheuristic approach (HMA) presented in [21]. Here, an initial population of solutions P_q for Q_q is generated and improved by HMA.

Finally, the uncoarsening phase successively selects and adds some previously extracted variables to the current problem Q_i $(0 < i < q)$, leading to a higher level (and larger) problem Q_{i-1}. The solutions P_i of the current problem together with the newly added variables are projected to the new problem Q_{i-1} and further optimized by HMA to obtain an improved population P_{i-1} of solutions. The uncoarsening phase stops when the highest level $i = 0$ is reached. At this point, the best solution found during the search is returned as the final solution to the problem Q_0.

The following sections detail each phase of our multilevel algorithm.

Algorithm 1. Outline of the backbone multilevel memetic algorithm for UBQP

```
1: Input: n_0 × n_0 matrix Q_0; maximum coarsening level q
2: Output: the best solution and its objective function value
3: i = 0
4: while i < q do
5:    Q_{i+1} ← Coarsen(Q_i) /* Create coarser intermediate matrices; see Section 2.2 */
6:    i = i + 1
7: end while
8: P_i ← Initial_Solution(Q_i) /* Generate initial solutions to the coarsest (lowest level)
   problem; see Section 2.3 */
9: P_i ← Memetic_Refinement(P_i, Q_i) /* Apply the memetic algorithm to optimize the
   initial solutions; see Section 2.4 */
10: while i > 0 do
11:    i = i − 1
12:    P_i ← Projection(P_{i+1}, Q_i) /* Back to a higher level matrix; see Section 2.5 */
13:    P_i ← Memetic_Refinement(P_i, Q_i) /* Apply the memetic algorithm to optimize the
       current solutions */
14: end while
```

2.2 The Backbone-Based Coarsening Phase

The backbone multilevel memetic algorithm employs a coarsening phase to cluster backbone variables, following the approach of our previous work [32]. The *backbone* terminology comes from the context of the satisfiability problem (SAT) [22,17]. There, the backbone of a satisfiable SAT problem is the set of literals which are true in every satisfying truth assignment. In our approach, we use a relaxed definition which is closely related to the notion of *strongly determined* and *consistent* variables explored in [10], identifying a backbone variable with regard to its contribution to a local optimum. In particular, the contribution of a variable x_k is defined as the change of the objective function value when x_k is flipped, i.e., changing the value of x_k to 1 - x_k.

From a given matrix Q_i $(i = 0, \ldots, q)$, our coarsening procedure repeats the following steps: 1) build a solution (an approximation of the global optimum) of problem Q_i, 2) use the solution to identify a set of backbone variables and, 3) create a simplified (or lower level) problem (i.e., a smaller matrix Q_{i+1}) by extracting from Q_i the rows and columns corresponding to the backbone variables. Algorithm 2 gives the pseudo-code of this backbone-based coarsening phase.

Algorithm 2. Pseudo-code of the backbone-based coarsening phase

Input: an $n_0 \times n_0$ matrix Q_0; maximum coarsening level q
Output: a series of coarser matrices $Q_1, Q_2, \ldots, Q_q$
$i = 0$
while $i < q$ **do**
  $S_i \leftarrow$ Initial_Solution(n_i)
  $S_i \leftarrow$ Tabu_Search(S_i, Q_i)
  Record the best solution S^* and its objective function value $f(S^*)$
  Identify the backbone variables B_i in level i with regard to the solution $S_i^{\#}$ /* Formula (2) */
  Remove the corresponding row and column of each variable in B_i from Q_i to get a lower level matrix Q_{i+1}
  $i = i + 1$
end while

The coarsening phase mainly consists of a while loop which starts from the highest level problem with $i = 0$. During the loop, we first construct an initial solution S_i by randomly assigning a value 0 or 1 to each variable of the current level problem and employ tabu search (see Section 2.4) to find a good local optimum for backbone identification. We additionally record the best solution S^* found so far and its objective function value $f(S^*)$.

To identify the set of backbone variables of Q_i, we use V_i to denote the set of the variables of Q_i and S_i a solution to Q_i. We apply the method proposed in [32] to first calculate, according to Equation (2), the contribution $VC_k(S_i^{\#})$ of each variable x_k in V_i with respect to the objective function f defined by formula (1), where $S_i^{\#}$ is a solution composed of S_i and the assignment of each backbone variable acquired prior to the level i.

$$VC_k(S_i^{\#}) = (1 - 2x_k)(Q_0(k,k) + \sum_{m \in N_0 \setminus \{k\}, x_m = 1} Q_0(k,m)) \tag{2}$$

where $N_0 = \{1, 2, \ldots, n_0\}$ and x_m is the value of each variable in $S_i^{\#}$. As noted in [11] and in a more general context in [13], $VC_k(S_i^{\#})$ identifies the change in $f(S_i^{\#})$ that results from changing the value of x_k to 1 - x_k. We observe that under a maximization objective if $S_i^{\#}$ is a locally optimal solution, then $VC_k(S_i^{\#}) \leq 0$ for all $k \in N_0$, and the current assignment of x_k will be more strongly determined as $VC_k(S_i^{\#})$ is more negative.

Then we use these $VC_k(S_i^{\#})$ values to sort the variables in a non-decreasing order and select the top na_i variables with respect to their contribution values. According to the study in [32], it is preferable to set $na_i = n_i \times 0.2$ if $i = 0$ and $na_i = na_{i-1} \times 0.4$ otherwise $(i > 0)$. These variables constitute the set of our

backbone variables denoted by B_i and are extracted from the matrix Q_i, leading to a new and simplified lower level problem Q_{i+1}.

Finally, we set $i = i + 1$ and repeat the while loop until i reaches the maximal level q (set to be equal to 3 in our experiments).

Obviously, each lower level problem Q_i $(i > 0)$ is a sub-problem of the highest level problem Q_0 and the solution of Q_i plus the value assignments of the backbone variables extracted prior to level i constitute a solution of Q_0.

2.3 Initial Population of Solutions

After the coarsening phase, a solution is sought for the problem of the lowest level (Q_q). For this, an initial population of solutions P_q is first constructed as follows. Each solution in P_q is generated in such a way that each variable receives randomly either 0 or 1. If this solution is not a duplicate of any solution in the population, it becomes a member of P_q. The above procedure repeats until the number of solutions reaches the population size which is fixed to 8 in this paper. The solutions are then optimized by applying the population-based memetic algorithm HMA which is explained below.

2.4 The Population-Based Memetic Algorithm HMA

The original population-based memetic algorithm HMA uses jointly the well-known uniform and a path-relinking crossover operators [21]. In this work, only the uniform crossover (UX) [30] is employed since experimental studies show that UX performs well under the multilevel framework. UX operates on two parent solutions randomly selected from the population and generates an offspring solution such that each of its variables takes the value of the corresponding variable in either parent one or parent two with equal probability.

For each offspring solution, HMA applies a tabu search procedure to improve the solution. The tabu search algorithm is based on a *one-flip move* neighborhood, consisting of changing (flipping) the value of a single variable x_i to its complementary value $1 - x_i$. The implementation of this neighborhood uses a fast incremental evaluation technique [12] to calculate the cost (move value) of transferring to each neighboring solution. Each time a move is carried out, the reverse move is forbidden for the next tl (tabu tenure) iterations. Accompanying this rule, a simple aspiration criterion is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than the current best solution. Tabu search stops when the best solution cannot be improved within a given number α of moves.

To maintain the diversity of its population, HMA uses a dedicated rule to decide whether an offspring solution is added to the population. For this, HMA introduces a quality-and-distance goodness score for the offspring solution with respect to the solutions of the population. If this goodness score is not smaller than that of the worst solution in the population, then the offspring solution is inserted into the population and replaces the worst solution. Otherwise, the worst solution is replaced by the offspring solution with a small probability. More details about the memetic algorithm can be found in [21].

2.5 The Asymmetric Uncoarsening Phase

In a multilevel approach, the uncoarsening phase carries out the inverse of the coarsening phase and typically traverses level by level the intermediate problems from the problems of the lowest level q to the highest level 0. For each level, each coarsened variable is uncoarsened to restore the original variables of the immediate upper level $i-1$. In this section, we explain how our uncoarsening phase is realized with regard to our backbone-based coarsening phase.

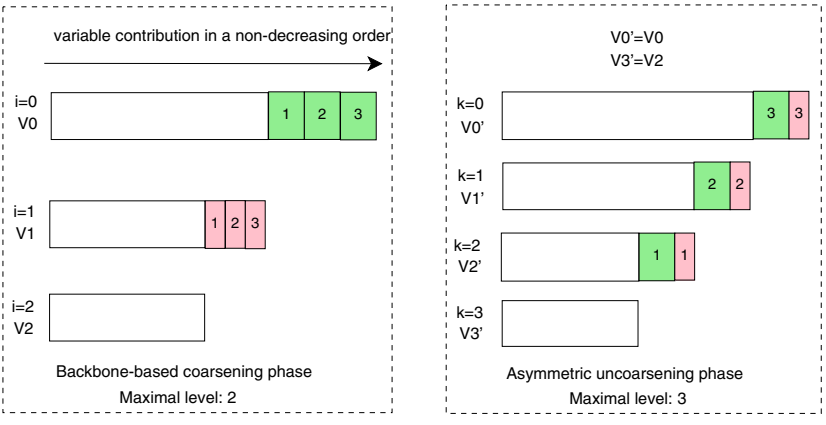

Fig. 1. Illustration of the asymmetric uncoarsening phase

Our uncoarsening phase progressively brings back the backbone variables extracted during the coarsening phase and allows them to take part in the subsequent optimizations. To achieve this, several strategies can be applied. For example, we can add back in a systematic way the extracted backbone variables in the strict reverse order of their extraction. We will discuss this systematic uncoarsening method in Section 4. Here we adopt another uncoarsening strategy (called asymmetric uncoarsening) which our experiments have shown to be more effective.

The idea of our asymmetric uncoarsening phase is based on the hypothesis that the values of the backbone variables with a high contribution (formula (2)) will have a higher probability of being optimal than the values of variables with a lower contribution. Therefore, it is desirable to freeze highly contributing variables at their assigned values as long as possible during the uncoarsening phase and to restore first those backbone variables with small contributions. These restored variables become part of the variables considered by the optimization process applied at each uncoarsening step. Since the backbone variables are restored according to contribution values instead of the order in which they are extracted, we refer to this strategy as an *asymmetric* uncoarsening phase. Notice that asymmetric uncoarsening may lead to a number of levels different from that created by the coarsening phase.

Figure 1 illustrates our asymmetric uncoarsening strategy. Each box represents the set V_i of all the variables of Q_i and the length of the box represents the size of V_i. The left portion of the figure shows a coarsening phase with 2 levels which extracts the backbone variables to simplify the highest level problem Q_0 into two lower level problems Q_1 and Q_2 in sequence. The right portion of the figure shows an asymmetric uncoarsening phase with 3 levels by adding back progressively the backbone variables from the lowest level problem $Q_3^{'}$ to a series of intermediate levels and finally to the highest level problem $Q_0^{'}$.

The process is achieved as follows. As mentioned in the backbone-based coarsening phase, the variables at each coarsening step are sorted in a non-decreasing order with regard to their contribution values and a certain number of variables are selected as backbone variables. Based on this, we separate the set of the backbone variables extracted at each coarsening step into K subsets, marked as $1, \ldots, K$ (In our example, $K = 3$, see below for the meaning of K). During the uncoarsening phase, we first select the subsets marked as 1 (which contain the backbone variables with small contributions) and add the variables contained in these subsets into set $V_3^{'}$ to create the set $V_2^{'}$, leading to the higher level problem $Q_2^{'}$. The same operations are successively applied to variable subsets marked as 2 and K (In our example, $K = 3$). In this way, we finally go back to the highest level problem Q_0.

Algorithm 3. Pseudo-code of the asymmetric uncoarsening phase

1: **Input**: The lowest problem Q_q, a fixed uncoarsening level $K > 1$
2: **Output**: The best binary n_0-vector S^* and the objective function value $f(S^*)$
3: Divide the set of backbone variables extracted at each coarsening level into K subsets with equal size
4: Fetch one subset from each coarsening level and combine them to generate the set UC_k for each uncoarsening level $k = K, \ldots, 1$
5: $k = K$
6: **while** $k > 0$ **do**
7: $k = k - 1$
8: Uncoarsen the variables in UC_{k+1} to obtain the matrix Q_k by inserting the row and column of each variable in UC_{k+1} into the matrix Q_{k+1}
9: Project each solution in population P_{k+1} to the corresponding solution in P_k
10: $P_k \leftarrow$ Memetic_Refinement(P_k, Q_k)
11: Record the best solution found so far S^* and its objective function $f(S^*)$
12: **end while**

The pseudo-code of the asymmetric uncoarsening phase is shown in Algorithm 3. To begin with, we separate the set of backbone variables extracted at each coarsening level into K subsets where K defines the number of the uncoarsening steps needed to go back to Q_0. Then we fetch one subset from each coarsening level and combine them to construct the set UC_k for each uncoarsening step k ($k = K, \ldots, 1$). This is a preparatory step for the uncoarsening phase (Alg. 3, lines 3-4).

From this point, an uncoarsening loop is launched with k starting at K. For each step, we reduce k by 1 and uncoarsen the variables in UC_{k+1} by including them into the set V_{k+1} to construct the set V_k and by inserting the row and column of each variable in UC_{k+1} into the matrix Q_{k+1} to obtain the matrix

Q_k. In addition, the solutions of population P_k are obtained by projecting the solutions of P_{k+1} plus the added backbone variables in UC_{k+1} with their corresponding values. Finally, the memetic optimization algorithm is used to refine the population P_k. The above loop continues until the highest level $k = 0$ is reached. The best solution found so far S^* and its objective function $f(S^*)$ are always recorded.

3 Experimental Results

In this section, we carry out extensive experiments to evaluate the performance of our backbone multilevel memetic algorithm (BMMA). Since the multilevel scheme is designed to cope with large problem instances, we take a set of 11 largest instances with variables from 5000 to 7000 that are known to be very difficult to solve for several algorithms. The source code of the generator and input files to replicate these problem instances can be found at: http://www.soften.ktu.lt/∼gintaras/ubqop_its.html. As indicated in [21,27,28], these instances are known to be much more challenging than those (with 2500 variables at most) from ORLIB [3]. Table 1 describes the properties of these benchmark instances including their sizes, densities and matrix coefficients. Note that the entry of each instance, say $Q(i, j)$ is a random integer number between -100 and +100. In addition, the best objective results ever reported in the literature are given in the last column (BKR).

Table 1. Main characteristics of Palubeckis benchmark test problems

Instance	n	Density	$Q(i,j)$	BKR
p5000.1	5000	0.5	[-100, +100]	8559680
p5000.2	5000	0.8	[-100, +100]	10836019
p5000.3	5000	0.8	[-100, +100]	10489137
p5000.4	5000	1.0	[-100, +100]	12252318
p5000.5	5000	1.0	[-100, +100]	12731803
p6000.1	6000	0.5	[-100, +100]	11384976
p6000.2	6000	0.8	[-100, +100]	14333855
p6000.3	6000	1.0	[-100, +100]	16132915
p7000.1	7000	0.5	[-100, +100]	14478676
p7000.2	7000	0.8	[-100, +100]	18249948
p7000.3	7000	1.0	[-100, +100]	20446407

Our BMMA algorithm is programmed in C and compiled using GNU GCC on a PC running Windows XP with Pentium 2.83GHz CPU and 2GB Memory. The stopping criteria is the completion of a round of the multilevel procedure rather than a time limit. Given the stochastic nature of our BMMA algorithm, each problem instance is independently solved 20 times.

Table 2 presents the results of our BMMA algorithm. Columns 1 and 2 give the instance names and the best known results in the literature. Columns 3 to 8 report respectively BMMA's best solution values *Best* and the number of times to reach *Best* over 20 runs in parentheses, the average solution values *Av.*, the standard deviation σ, the best time T_{best} and the average time T_{b_avr} to reach the best solution values *Best*, and the average time T_{AVR} consumed for a BMMA

Table 2. Computational results of the BMMA algorithm

Instance	BKR	BMMA					
		Best	Av.	σ	T_{best}	T_{b_avr}	T_{AVR}
p5000.1	8559680	8559680(1)	8558912	424	86	86	645
p5000.2	10836019	10836019(2)	10835253	527	92	219	607
p5000.3	10489137	10489137(2)	10488450	1057	344	351	630
p5000.4	12252318	12252318(2)	12251122	809	98	275	584
p5000.5	12731803	12731803(11)	12731423	493	158	326	554
p6000.1	11384976	11384976(5)	11384566	854	170	400	878
p6000.2	14333855	14333855(5)	14333101	1132	341	416	939
p6000.3	16132915	16132915(3)	16130610	1147	179	545	848
p7000.1	14478676	14478676(4)	14477235	1423	656	944	1349
p7000.2	18249948	18249948(1)	18247518	1424	951	951	1289
p7000.3	20446407	20446407(9)	20444603	3414	550	761	1132
Av.	13626885	13626885	13625708	1155	330	479	860
Deviation%.		0.000000	0.008633				

run (in seconds). The last two rows report the average over the 11 instances for each evaluation criteria and the average percent deviation of the solution values from the best known values.

From Table 2, we find that the average objective values attained by BMMA are very close to the best known results, with an average percent deviation 0.008633%. Finally, the best and average time to reach our best solution values are only 330 and 479 seconds, respectively. In sum, our BMMA algorithm is quite effective in finding the best known values for these challenging instances.

3.1 Comparison between the BMMA and HMA Algorithms

We now assess the advantage of the multilevel scheme by comparing the BMMA algorithm with its optimization algorithm HMA which is applied at each uncoarsening level (see Section 2.4). For this purpose, we run HMA within the time limit T_{AVR} (see Table 2), i.e., the time of a BMMA run. The results are shown in Table 3.

From Tables 2 and 3, one observes that the BMMA algorithm outperforms the HMA algorithm in terms of several different criteria. Specifically, when it comes to the best solution values found, HMA is inferior to BMMA on 3 instances

Table 3. Computational results of the HMA algorithm

Instance	BKR	HMA					
		Best	Av.	σ	T_{best}	T_{b_avr}	T_{AVR}
p5000.1	8559680	8559355(1)	8558671	783	349	349	600
p5000.2	10836019	10836019(1)	10835298	262	452	452	600
p5000.3	10489137	10489137(2)	10488711	637	518	555	600
p5000.4	12252318	12252275(1)	12250982	637	589	589	600
p5000.5	12731803	12731803(9)	12731195	684	251	434	600
p6000.1	11384976	11384807(1)	11384506	812	884	884	900
p6000.2	14333855	14333855(1)	14332723	1456	761	761	900
p6000.3	16132915	16132915(2)	16130419	1098	603	641	900
p7000.1	14478676	14478676(1)	14476628	1300	1072	1072	1300
p7000.2	18249948	18249948(2)	18247600	1403	1086	1119	1300
p7000.3	20446407	20446407(6)	20444120	3728	508	855	1300
Av.	13626885	13626836	13625532	1164	643	701	873
Deviation%.		0.000358	0.009928				

(5000.1, 5000.4 and 6000.1). In addition, HMA's best and average solution deviation from the best known results are 0.000358% and 0.009928%, in comparison with BMMA's deviation values 0.000000% and 0.008633%. Furthermore, the best and average time for BMMA to find the best solution values are respectively 330 and 479 seconds which are 49% and 32% less than that of HMA. These outcomes must be qualified by observing that, as shown in [21], given longer time limits HMA consistently attains the best-known results of the literature.

3.2 Comparison between BMMA and Other State-of-the-Art Algorithms

In order to further evaluate our BMMA algorithm, we compare it with several best-performing algorithms in the literature. These methods are respectively named ITS [28], MST2 [27], SA [16] D^2TS [13], HMA [21], BGTS [33] and DHNN-EDA [34]. Given the fact that all these algorithms were run under different environments, often with larger time limits, it is thus hard to make a completely fair comparison. Nevertheless, this experiment indicates that our proposed algorithm performs exceedingly well in relation to these reference state-of-the-art algorithms.

Table 4 compares the best solution values reported by each reference algorithm. To highlight the difference among the reference algorithms, we show the gap between the best solution of each algorithm and the best known solution. From Table 4, we observe that the BMMA, BGTS and HMA algorithms perform similarly well in that they are all able to attain the best known results on all the instances. In addition, the BMMA algorithm outperforms the other four reference algorithms, named ITS, MST2, SA and DHNN-EDA and is slightly better than the D^2TS algorithm. To be specific, the four reference algorithms have an average solution gap from 586 to 2661 and the D^2TS algorithm has an average solution gap of 39 to the best known values.

Table 5 compares the average time to reach the best solution values. The BGTS, HMA and D^2TS algorithms are run on a PC with a Pentium 2.66GHz CPU and DHNN-EDA is run on a comparable PC with a Pentium 2.8GHz

Table 4. Comparison between BMMA and other algorithms : Gap to the best known solution

Instance	BMMA	BGTS	D^2TS	HMA	ITS	MST2	SA	DHNN-EDA
p5000.1	0	0	325	0	700	325	1432	2244
p5000.2	0	0	0	0	0	582	582	1576
p5000.3	0	0	0	0	0	0	354	813
p5000.4	0	0	0	0	934	1643	444	1748
p5000.5	0	0	0	0	0	0	1025	1655
p6000.1	0	0	0	0	0	0	430	453
p6000.2	0	0	0	0	88	0	675	4329
p6000.3	0	0	0	0	2729	0	0	4464
p7000.1	0	0	0	0	340	1607	2579	4529
p7000.2	0	0	104	0	1651	2330	5552	5750
p7000.3	0	0	0	0	0	0	2264	1707
Av.	0	0	39	0	586	589	1394	2661

Table 5. Comparison between BMMA and other algorithms : Best time (seconds)

Instance	BMMA	BGTS	D^2TS	HMA	ITS	MST2	SA	DHNN-EDA
p5000.1	86	556	2855	587	507	540	605	1572
p5000.2	219	1129	1155	464	421	649	691	1572
p5000.3	351	874	1326	758	672	788	945	1572
p5000.4	275	379	838	1453	596	935	1059	1572
p5000.5	326	629	623	686	551	719	1057	1572
p6000.1	400	597	509	994	978	1037	615	2378
p6000.2	416	428	1543	1332	839	887	1085	2378
p6000.3	545	601	2088	1406	957	1218	1474	2378
p7000.1	944	1836	1217	1435	1771	1449	1952	3216
p7000.2	951	1569	849	1770	1013	1722	1738	3216
p7000.3	761	703	3520	2456	1446	2114	2138	3216
Av.	479	846	1502	1213	886	1096	1214	2240

CPU. The ITS, MST2 and SA algorithms are run on a Pentium III 800 PC. We transformed their original times by dividing them by 3 given that our computer is about 3 times faster than the Pentium III 800 PC [13].

From Table 5, we can make the following observations. First, among the three algorithms (BMMA, BGTS and HMA) which reach the best known results for all the 11 instances, our proposed BMMA algorithm needs an average time of 479 seconds to reach the best solution values, against 846 and 1213 seconds for the BGTS and HMA algorithms respectively.

Second, for the 4 other algorithms (D^2TS, ITS, MST2, SA, DHNN-EDA) which fail to find the best known solutions for at least two instances, our BMMA algorithm clearly dominates all of them both in terms of the best solution values and computational efficiency. In particular, BMMA needs one fifth of the time needed by the most recent DHNN-EDA algorithm to attain much better solutions.

In sum, this experimental study demonstrates the merit of our BMMA algorithm for solving the large instances of the UBQP problem.

4 Discussion

In order to verify the proposed asymmetric backbone uncoarsening phase indeed works well compared to a more customary type of multilevel procedure, we also implemented a symmetric backbone uncoarsening phase, which adds back progressively the backbone variables from the lowest level Q_q to the highest level Q_0 by following the strict reverse order the backbone variables are extracted during the coarsening phase. For this experiment, we kept other components of our BMMA algorithm unchanged except the uncoarsening component. Table 6 shows the computational results of the two different uncoarsening methods.

As we can see in Table 6, the asymmetric uncoarsening performs better than the symmetric one in terms of the best, average and standard deviation values. Specifically, the asymmetric uncoarsening obtains the best known values for all the instances while the symmetric uncoarsening leads only to 6 best known results. Moreover, the asymmetric uncoarsening reaches better average values with a smaller deviation from the best known results (0.008633% versus 0.014415%

Table 6. Comparison between the symmetric and asymmetric uncoarsening methods

Instance	BKR	Symmetric			Asymmetric		
		Best	Av.	σ	Best	Av.	σ
p5000.1	8559680	8559075	8558510	**412**	**8559680**	**8558912**	424
p5000.2	10836019	**10836019**	10834954	707	**10836019**	**10835253**	**527**
p5000.3	10489137	**10489137**	10487669	1247	**10489137**	**10488450**	**1057**
p5000.4	12252318	**12252318**	12250980	**662**	**12252318**	**12251122**	809
p5000.5	12731803	**12731803**	12731247	525	**12731803**	**12731423**	**493**
p6000.1	11384976	11384733	11384026	1285	**11384976**	**11384566**	**854**
p6000.2	14333855	14333727	14332568	**997**	**14333855**	**14333101**	1132
p6000.3	16132915	16130915	16129770	**683**	**16132915**	**16130610**	1147
p7000.1	14478676	**14478676**	14475669	**1344**	**14478676**	**14477235**	1423
p7000.2	18249948	18249844	18246763	1513	**18249948**	**18247518**	**1424**
p7000.3	20446407	**20446407**	20441970	3971	**20446407**	**20444603**	**3414**
Av.	13626885	13626605	13624921	1213	**13626885**	**13625708**	**1155**
Deviation%.	–	0.002055	0.014415	–	**0.000000**	**0.008633**	–

for symmetric uncoarsening). In addition, the asymmetric uncoarsening is also superior to the symmetric uncoarsening in terms of the standard deviation, with the value 1155 versus 1213.

5 Conclusion

Solving large random UBQP instances is a challenging task. In this paper, we have shown the multilevel approach constitutes an effective approach to cope with these large random UBQP instances. The proposed algorithm combines a backbone-based coarsening phase, an asymmetric uncoarsening phase and a memetic refinement procedure, each incorporating tabu search to obtain improved solutions. Experiments on the most challenging instances (with 5000 to 7000 variables) from the literature demonstrate that the proposed algorithm is able to find all the best known results while using much less computing time than the previous state-of-the-art algorithms. We anticipate that our approach can be further refined by investigating alternative strategies for the coarsening and uncoarsening phases.

Acknowledgment. We would like to thank the anonymous referees for their helpful comments and suggestions. The work is partially supported by the Pays de la Loire Region (France) within the RaDaPop (2009-2013) and LigeRO (2010-2013) projects.

References

1. Alidaee, B., Kochenberger, G.A., Ahmadian, A.: 0-1 quadratic programming approach for the optimal solution of two scheduling problems. International Journal of Systems Science 25, 401–408 (1994)
2. Amini, M., Alidaee, B., Kochenberger, G.A.: A scatter search approach to unconstrained quadratic binary programs. In: New Methods in Optimization, pp. 317–330. McGraw-Hill, New York (1999)
3. Beasley, J.E.: Obtaining test problems via internet. Journal of Global Optimization 8, 429–433 (1996)

4. Beasley, J.E.: Heuristic algorithms for the unconstrained binary quadratic programming problem. Working Paper, The Management School, Imperial College, London, England (1998)
5. Benlic, U., Hao, J.K.: A Multilevel Memetic Approach for Improving Graph k-Partitions. IEEE Transactions on Evolutionary Computation 15(5), 624–642 (2011)
6. Borgulya, I.: An evolutionary algorithm for the binary quadratic problems. Advances in Soft Computing 2, 3–6 (2005)
7. Boros, E., Hammer, P.L., Sun, R., Tavares, G.: A max-flow approach to improved lower bounds for quadratic 0-1 minimization. Discrete Optimization 5(2), 501–529 (2008)
8. Boros, E., Hammer, P.L., Tavares, G.: Local search heuristics for Quadratic Unconstrained Binary Optimization (QUBO). Journal of Heuristics 13, 99–132 (2007)
9. Chardaire, P., Sutter, A.: A decomposition method for quadratic zero-one programming. Management Science 41(4), 704–712 (1994)
10. Glover, F.: Heuristics for integer programming using surrogate constraints. Decision Sciences 8(1), 156–166 (1977)
11. Glover, F., Kochenberger, G.A., Alidaee, B.: Adaptive memory tabu search for binary quadratic programs. Management Science 44, 336–345 (1998)
12. Glover, F., Hao, J.K.: Efficient Evaluation for Solving 0-1 Unconstrained Quadratic Optimization Problems. International Journal of Metaheuristics 1(1), 3–10 (2010)
13. Glover, F., Lü, Z., Hao, J.K.: Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR: A Quarterly Journal of Operations Research 8(3), 239–253 (2010)
14. Harary, F.: On the notion of balanced of a signed graph. Michigan Mathematical Journal 2, 143–146 (1953)
15. Helmberg, C., Rendl, F.: Solving quadratic (0,1)-problem by semidefinite programs and cutting planes. Mathematical Programming 82, 388–399 (1998)
16. Katayama, K., Narihisa, H.: Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. European Journal of Operational Research 134, 103–119 (2001)
17. Kilby, P., Slaney, J.K., Thiebaux, S., Walsh, T.: Backbones and backdoors in satisablity. In: Proceedings of AAAI 2005, pp. 1368–1373 (2005)
18. Kochenberger, G.A., Glover, F., Alidaee, B., Rego, C.: A unified modeling and solution framework for combinatorial optimization problems. OR Spectrum 26, 237–250 (2004)
19. Krarup, J., Pruzan, A.: Computer aided layout design. Mathematical Programming Study 9, 75–84 (1978)
20. Lodi, A., Allemand, K., Liebling, T.M.: An evolutionary heuristic for quadratic 0-1 programming. European Journal of Operational Research 119(3), 662–670 (1999)
21. Lü, Z., Glover, F., Hao, J.K.: A hybrid metaheuristic approach to solving the UBQP problem. European Journal of Operational Research 207(3), 1254–1262 (2010)
22. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining computational complexity for characteristic phase transitions. Nature 400, 133–137 (1998)
23. McBride, R.D., Yormark, J.S.: An implicit enumeration algorithm for quadratic integer programming. Management Science 26, 282–296 (1980)
24. Meyerhenke, H., Monien, B., Sauerwald, T.: A New Diffusion-based Multilevel Algorithm for Computing Graph Partitions of Very High Quality. Journal of Parallel and Distributed Computing 69(9), 750–761 (2009)

25. Merz, P., Freisleben, B.: Genetic algorithms for binary quadratic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 417–424. Morgan Kaufmann (1999)
26. Merz, P., Katayama, K.: Memetic algorithms for the unconstrained binary quadratic programming problem. BioSystems 78, 99–118 (2004)
27. Palubeckis, G.: Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. Annals of Operations Research 131, 259–282 (2004)
28. Palubeckis, G.: Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica 17(2), 279–296 (2006)
29. Pardalos, P., Rodgers, G.P.: Computational aspects of a branch and bound algorithm for quadratic zero-one programming. Computing 45, 131–144 (1990)
30. Syswerda, G.: Uniform crossover in genetic algorithms. In: Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 2–9 (1989)
31. Toulouse, M., Thulasiraman, K., Glover, F.: Multi-level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning. In: Amestoy, P.R., Berger, P., Daydé, M., Duff, I.S., Frayssé, V., Giraud, L., Ruiz, D. (eds.) Euro-Par 1999. LNCS, vol. 1685, pp. 533–542. Springer, Heidelberg (1999)
32. Wang, Y., Lü, Z., Glover, F., Hao, J.K.: Backbone guided Tabu Search for solving the UBQP problem. Journal of Heuristics (2011), doi:10.1007/s10732-011-9164-4
33. Wang, Y., Lü, Z., Glover, F., Hao, J.-K.: Effective Variable Fixing and Scoring Strategies for Binary Quadratic Programming. In: Hao, J.-K. (ed.) EvoCOP 2011. LNCS, vol. 6622, pp. 72–83. Springer, Heidelberg (2011)
34. Wang, J., Zhou, Y., Yin, J.: Combining tabu hopfield network and estimation of distribution for unconstrained binary quadratic programming problem. Expert System With Applications (2011), doi:10.1016/j.eswa.2011.05060
35. Walshaw, C., Cross, M.: Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm. SIAM Journal on Scientific Computing 22(1), 63–80 (2000)
36. Walshaw, C.: Multilevel refinement for combinatorial optimisation problems. Annals of Operations Research 131, 325–372 (2004)